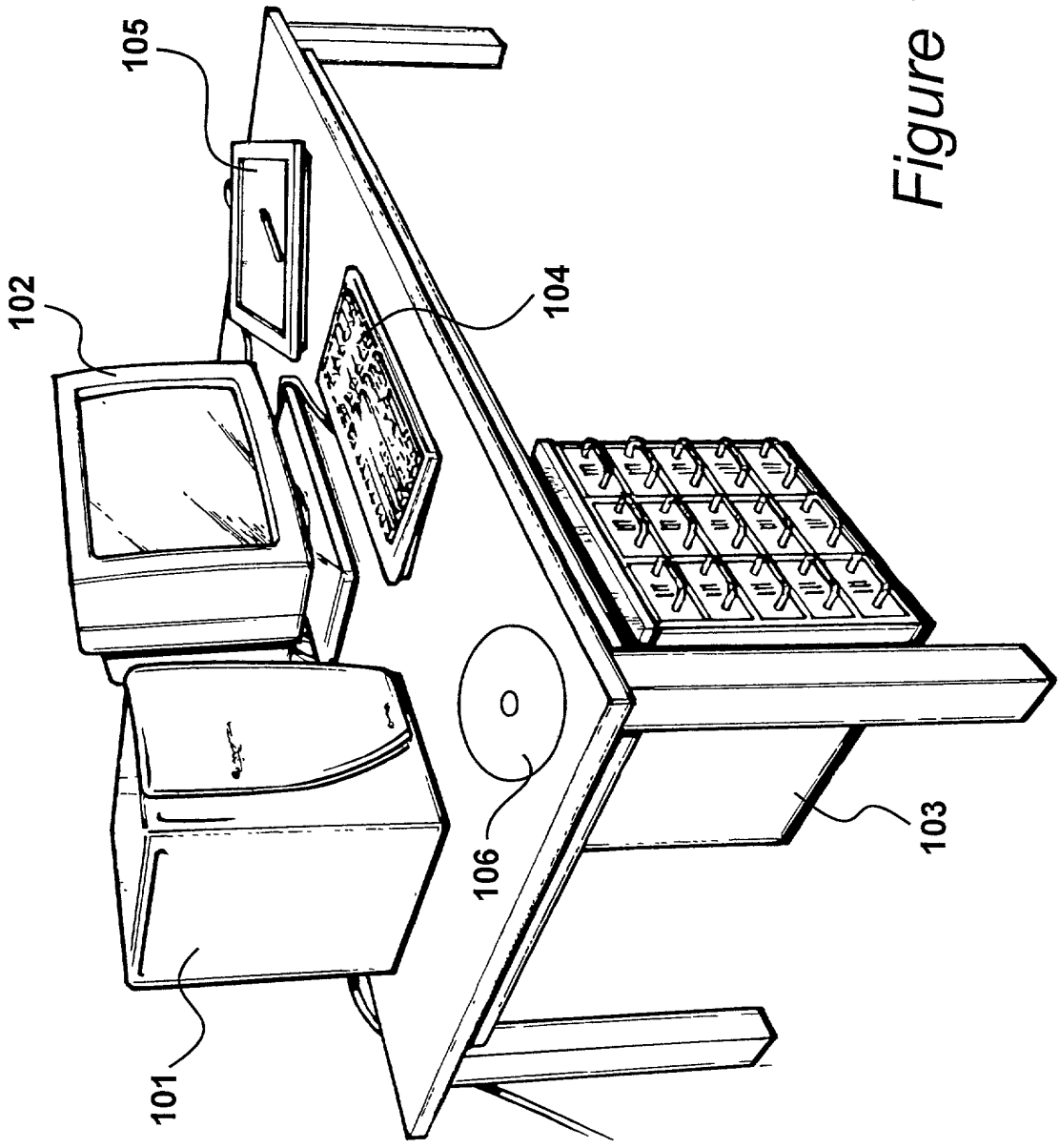


Figure 1

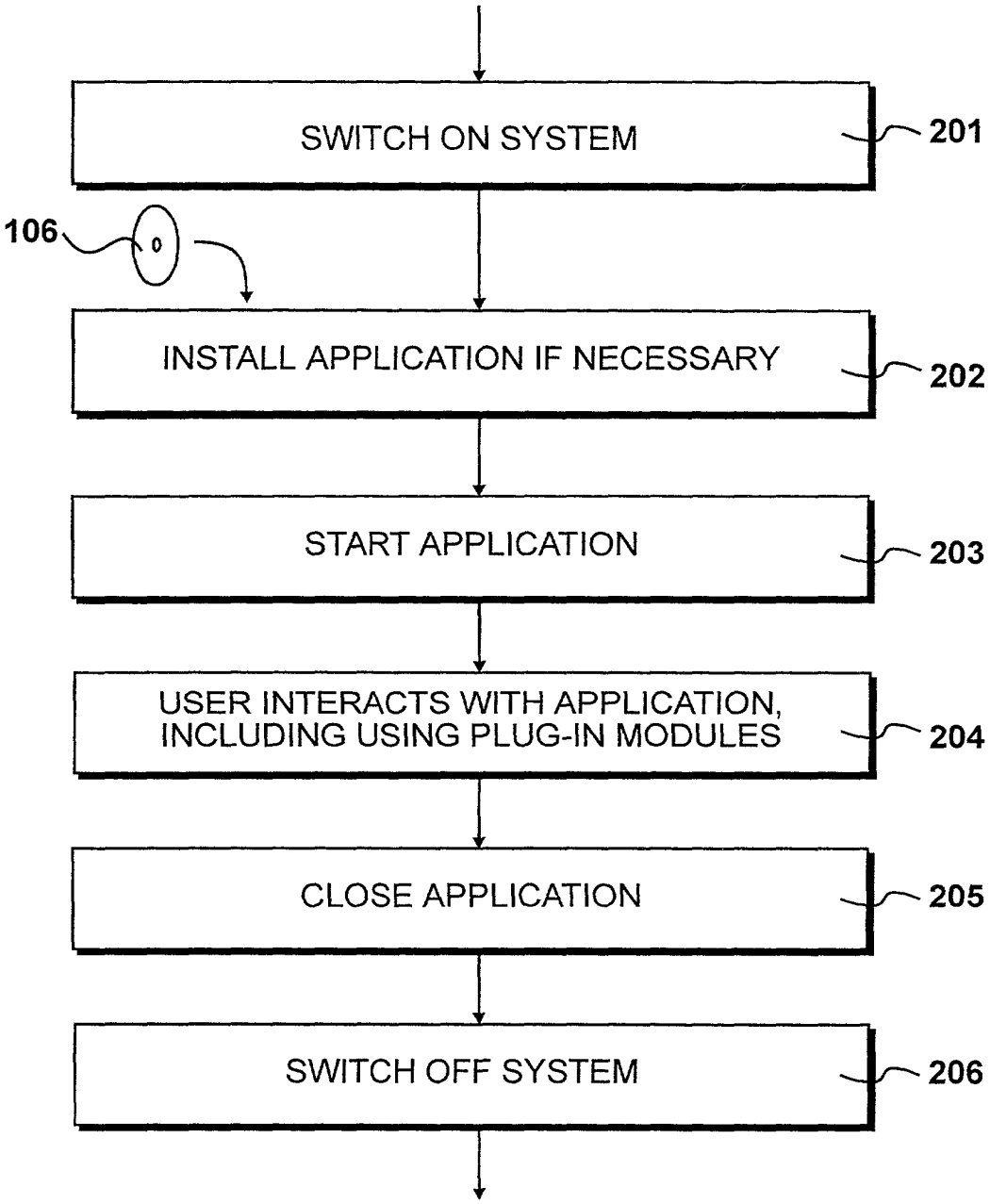

SWITCH ON SYSTEM
201
106
INSTALL APPLICATION IF NECESSARY
202
START APPLICATION
203
USER INTERACTS WITH APPLICATION, INCLUDING USING PLUG-IN MODULES
204
CLOSE APPLICATION
205
SWITCH OFF SYSTEM
206

Figure 2

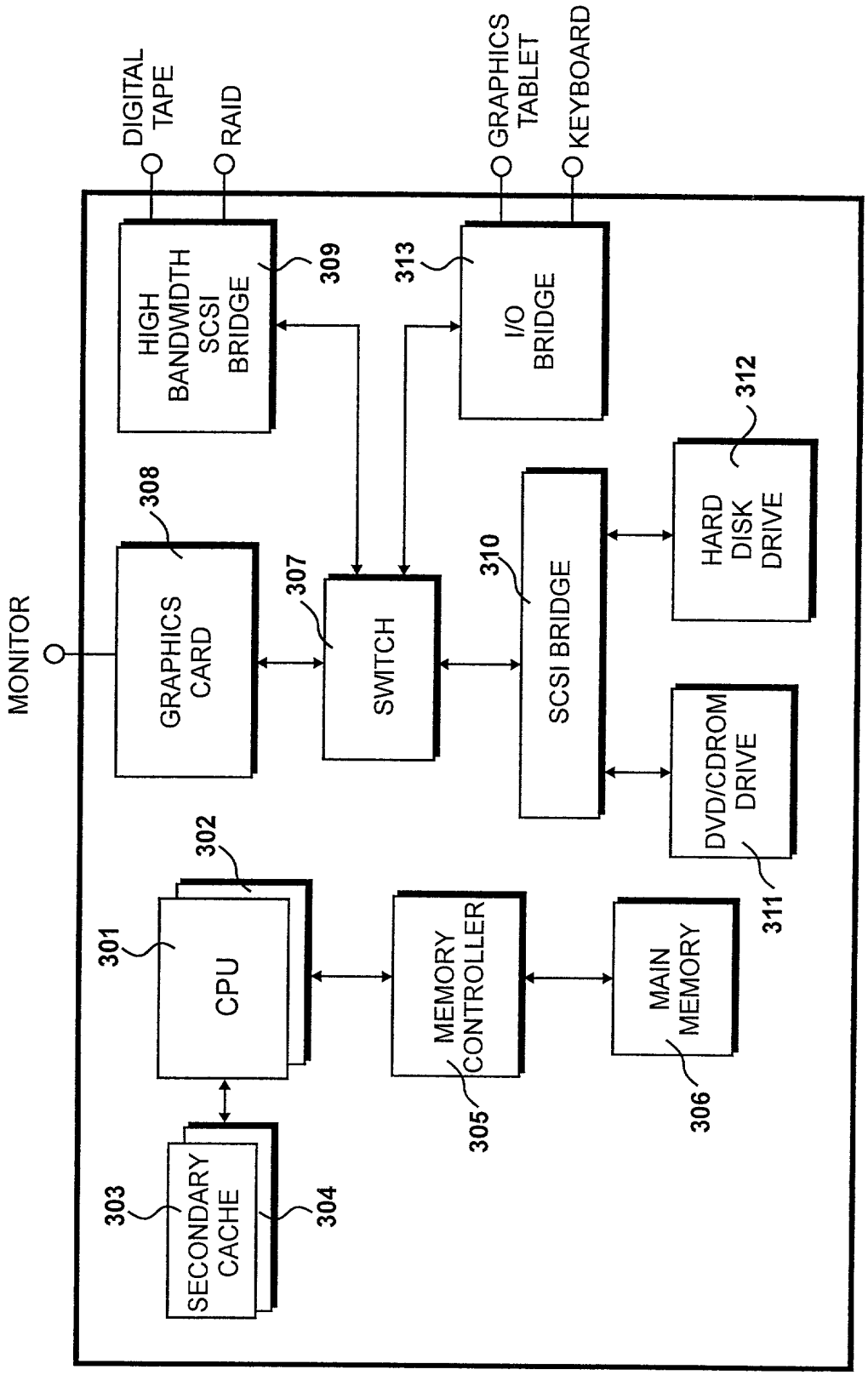
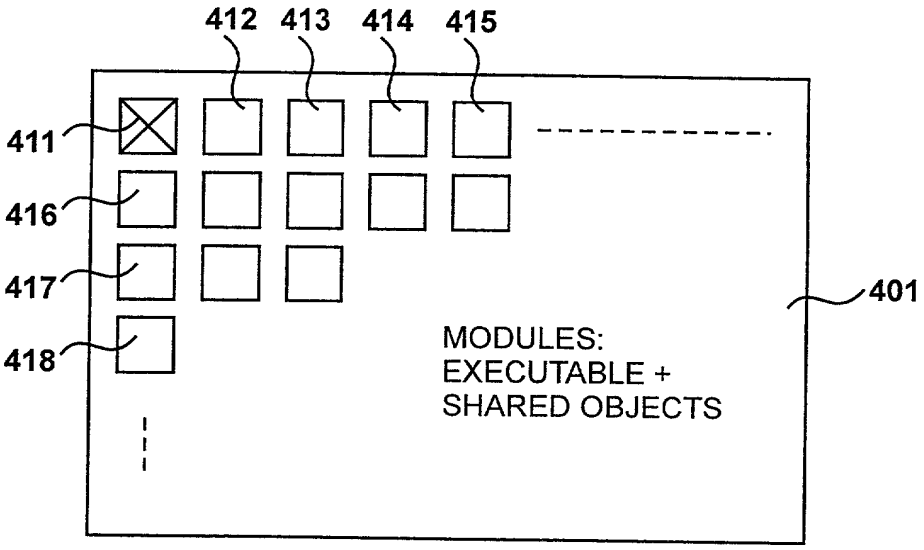
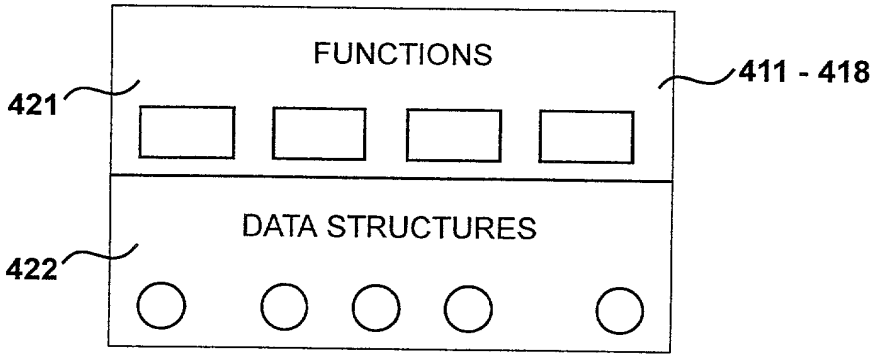

DIGITAL TAPE
RAID
GRAPHICS TABLET
KEYBOARD
HIGH BANDWDTH SCSI BRIDGE
309
313
I/O BRIDGE
312
HARD DISK DRIVE
MONITOR
308
GRAPHICS CARD
307
SWITCH
310
SCSI BRIDGE
DVD/CDROM DRIVE
311
302
301
CPU
MEMORY CONTROLLER
MAIN MEMORY
306
305
303
SECONDARY CACHE
304

Figure 3


412
413
414
415
411
416
417
418
401
MODULES:
EXECUTABLE +
SHARED OBJECTS


FUNCTIONS
411 - 418
421
DATA STRUCTURES
422

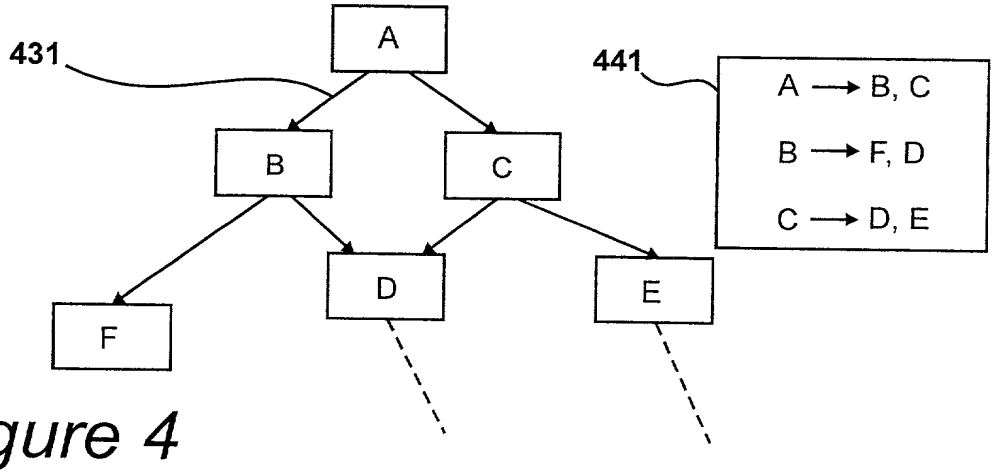

431
A
B
C
D
E
F
441
A → B, C
B → F, D
C → D, E

Figure 4

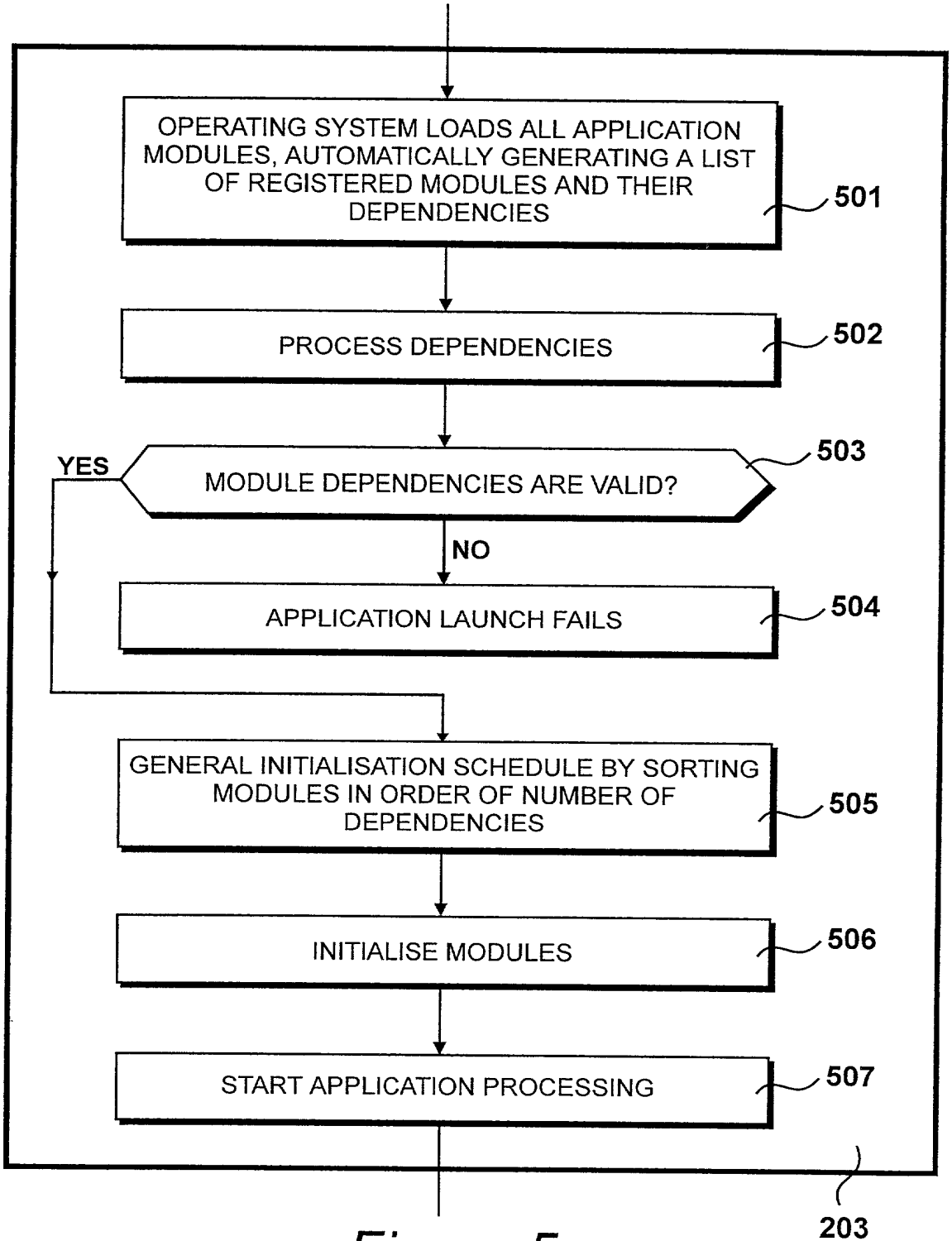

OPERATING SYSTEM LOADS ALL APPLICATION MODULES, AUTOMATICALLY GENERATING A LIST OF REGISTERED MODULES AND THEIR DEPENDENCIES
501
PROCESS DEPENDENCIES
502
YES
MODULE DEPENDENCIES ARE VALID?
503
NO
APPLICATION LAUNCH FAILS
504
GENERAL INITIALISATION SCHEDULE BY SORTING MODULES IN ORDER OF NUMBER OF DEPENDENCIES
505
INITIALISE MODULES
506
START APPLICATION PROCESSING
507
203

Figure 5

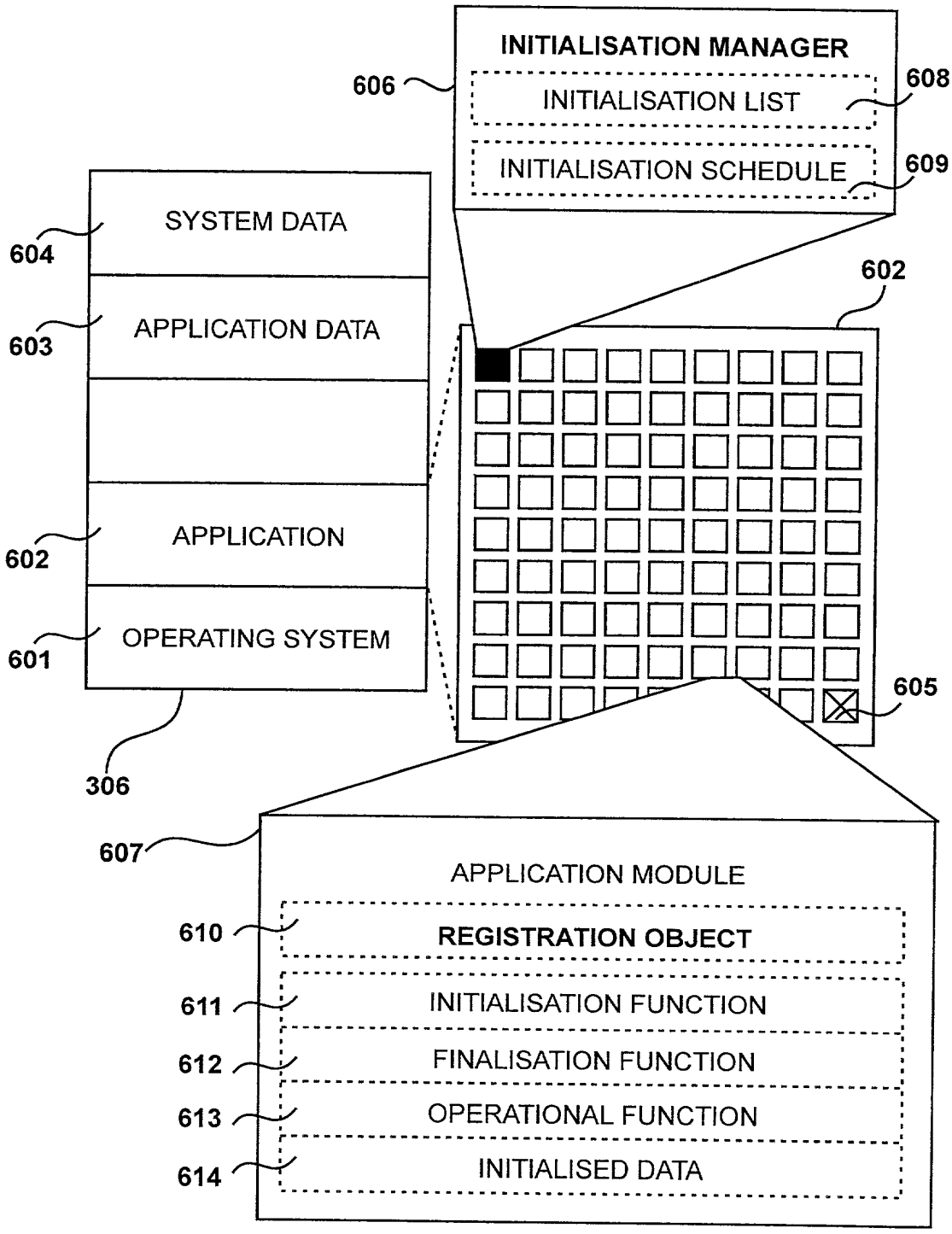

INITIALISATION MANAGER
606
608
INITIALISATION LIST
INITIALISATION SCHEDULE
609
SYSTEM DATA
604
APPLICATION DATA
603
602
APPLICATION
602
OPERATING SYSTEM
601
605
306
607
APPLICATION MODULE
610
REGISTRATION OBJECT
611
INITIALISATION FUNCTION
612
FINALISATION FUNCTION
613
OPERATIONAL FUNCTION
614
INITIALISED DATA

Figure 6

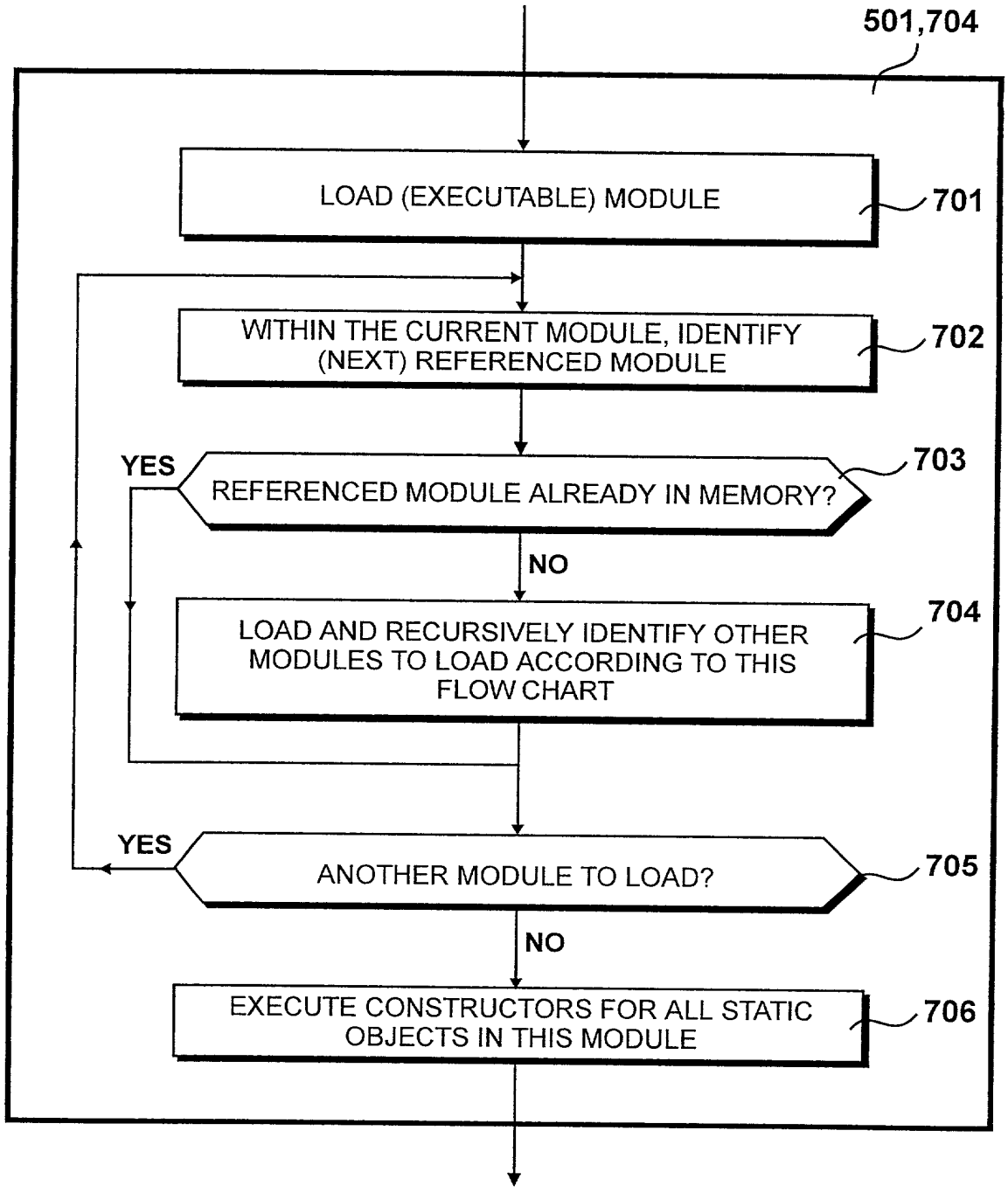

501,704
LOAD (EXECUTABLE) MODULE
701
WITHIN THE CURRENT MODULE, IDENTIFY (NEXT) REFERENCED MODULE
702
YES
REFERENCED MODULE ALREADY IN MEMORY?
703
NO
LOAD AND RECURSIVELY IDENTIFY OTHER MODULES TO LOAD ACCORDING TO THIS FLOW CHART
704
YES
ANOTHER MODULE TO LOAD?
705
NO
EXECUTE CONSTRUCTORS FOR ALL STATIC OBJECTS IN THIS MODULE
706

Figure 7

```
// Filename: foo.cpp

#include <database.h>
#include <osal.h>
#include <initialize.h>

#include "foo.h"

// Static objects                                    801

InInit<foo> _instance;    //registration object

// Constructor for registration object

InInit<foo>::InInit () {                             802

// Registering Dependencies
   addDependency(InInit<database>::getType());actual call pattern;
   registerDependency(typeInfo(InInit<osal>).typeID());
   registerDependency(typeInfo(InInit<initialize>).typeID());


}

   Module initialisation and finalisation functions
   InInit<foo>::performInitialise()
               () {                                  803

   // whatever initialisation this module requires

}
  void InInit<foo>::performFinalize()
                                                     804
        () {

   // whatever finalisation this module requires

}

// All other module functions from this point on …
```

Figure 8

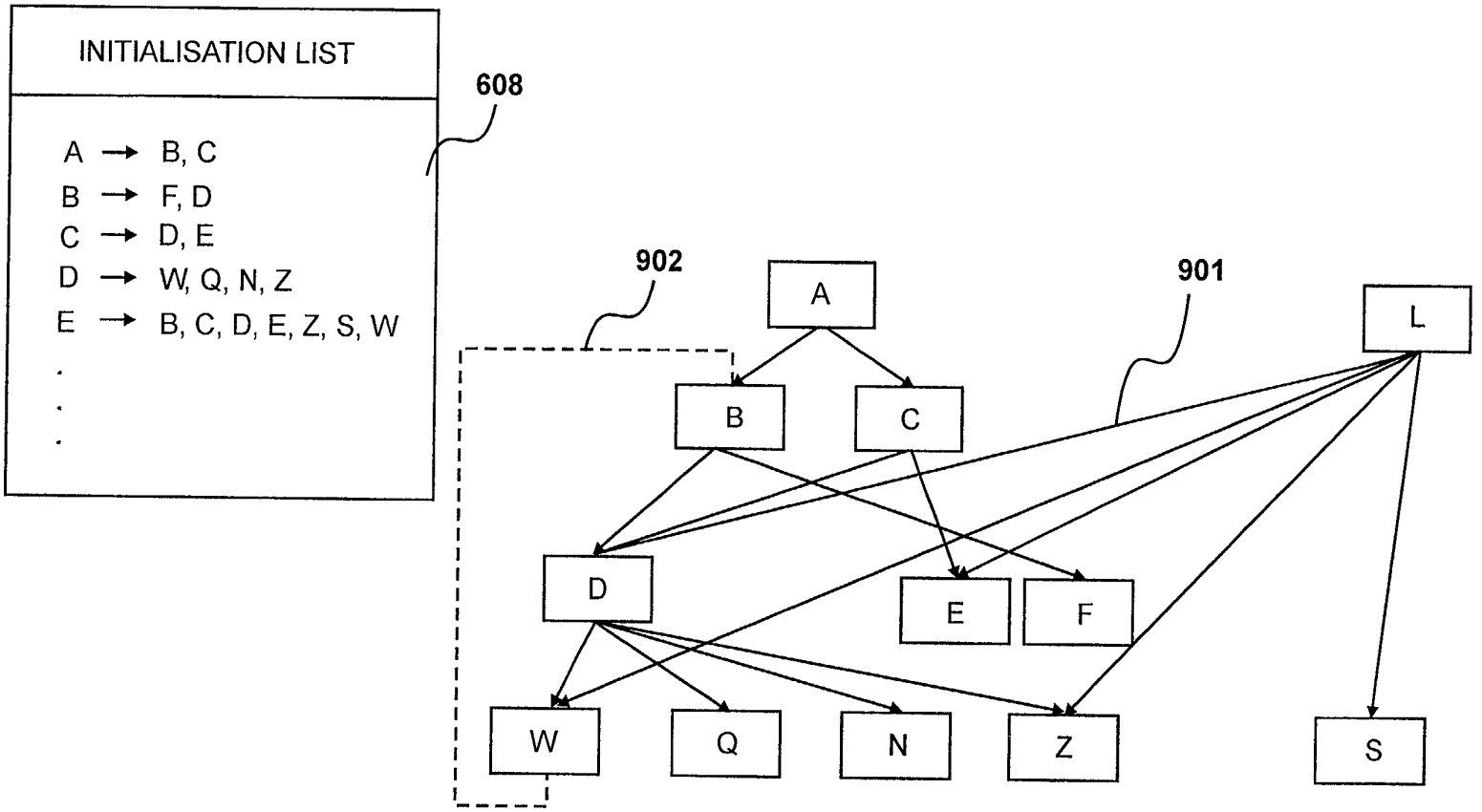

INITIALISATION LIST
608
A → B, C
B → F, D
C → D, E
D → W, Q, N, Z
E → B, C, D, E, Z, S, W
902
901
A
B
C
D
E
F
L
W
Q
N
Z
S

Figure 9

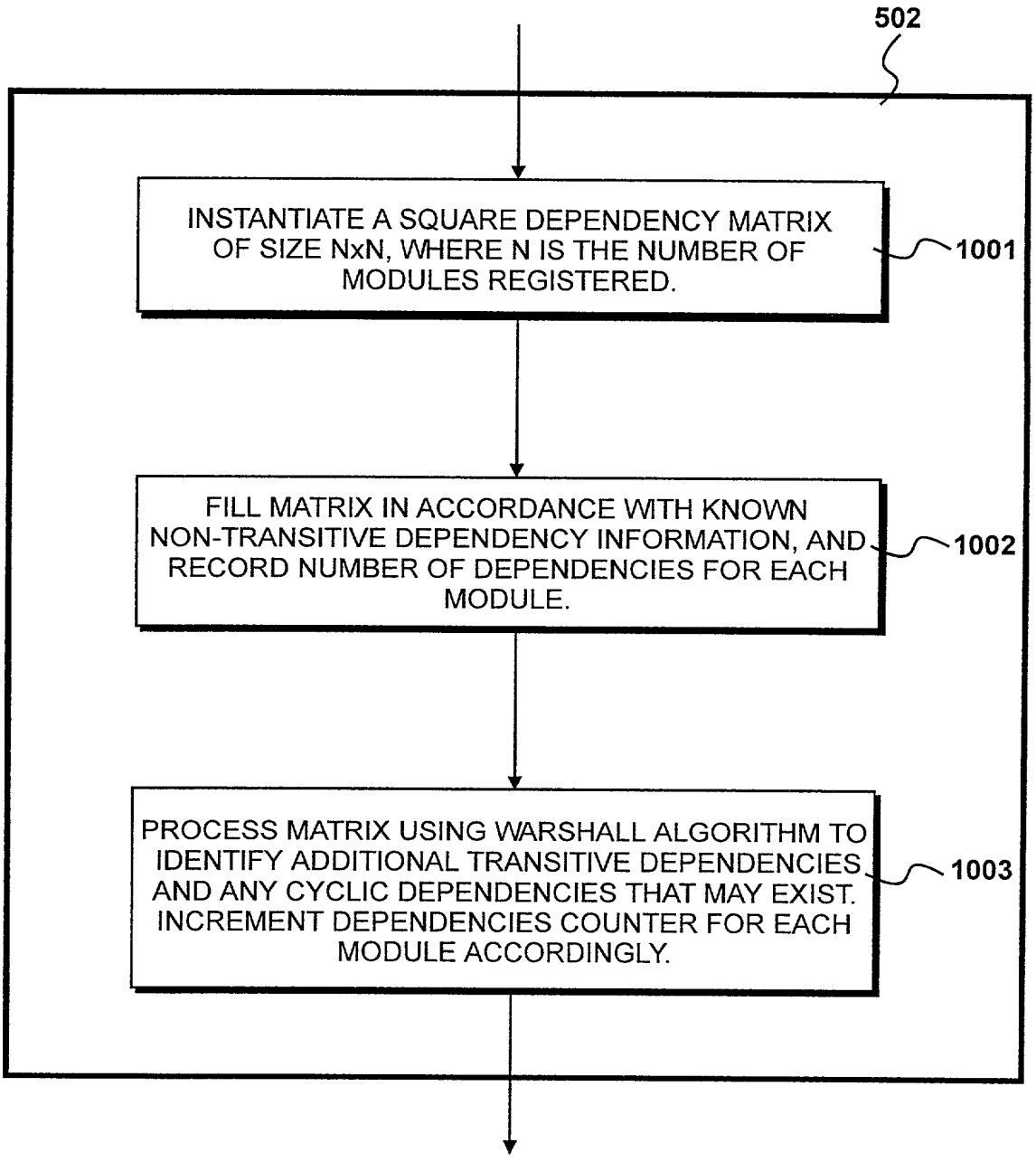

502
INSTANTIATE A SQUARE DEPENDENCY MATRIX OF SIZE NxN, WHERE N IS THE NUMBER OF MODULES REGISTERED.
1001
FILL MATRIX IN ACCORDANCE WITH KNOWN NON-TRANSITIVE DEPENDENCY INFORMATION, AND RECORD NUMBER OF DEPENDENCIES FOR EACH MODULE.
1002
PROCESS MATRIX USING WARSHALL ALGORITHM TO IDENTIFY ADDITIONAL TRANSITIVE DEPENDENCIES AND ANY CYCLIC DEPENDENCIES THAT MAY EXIST. INCREMENT DEPENDENCIES COUNTER FOR EACH MODULE ACCORDINGLY.
1003

Figure 10

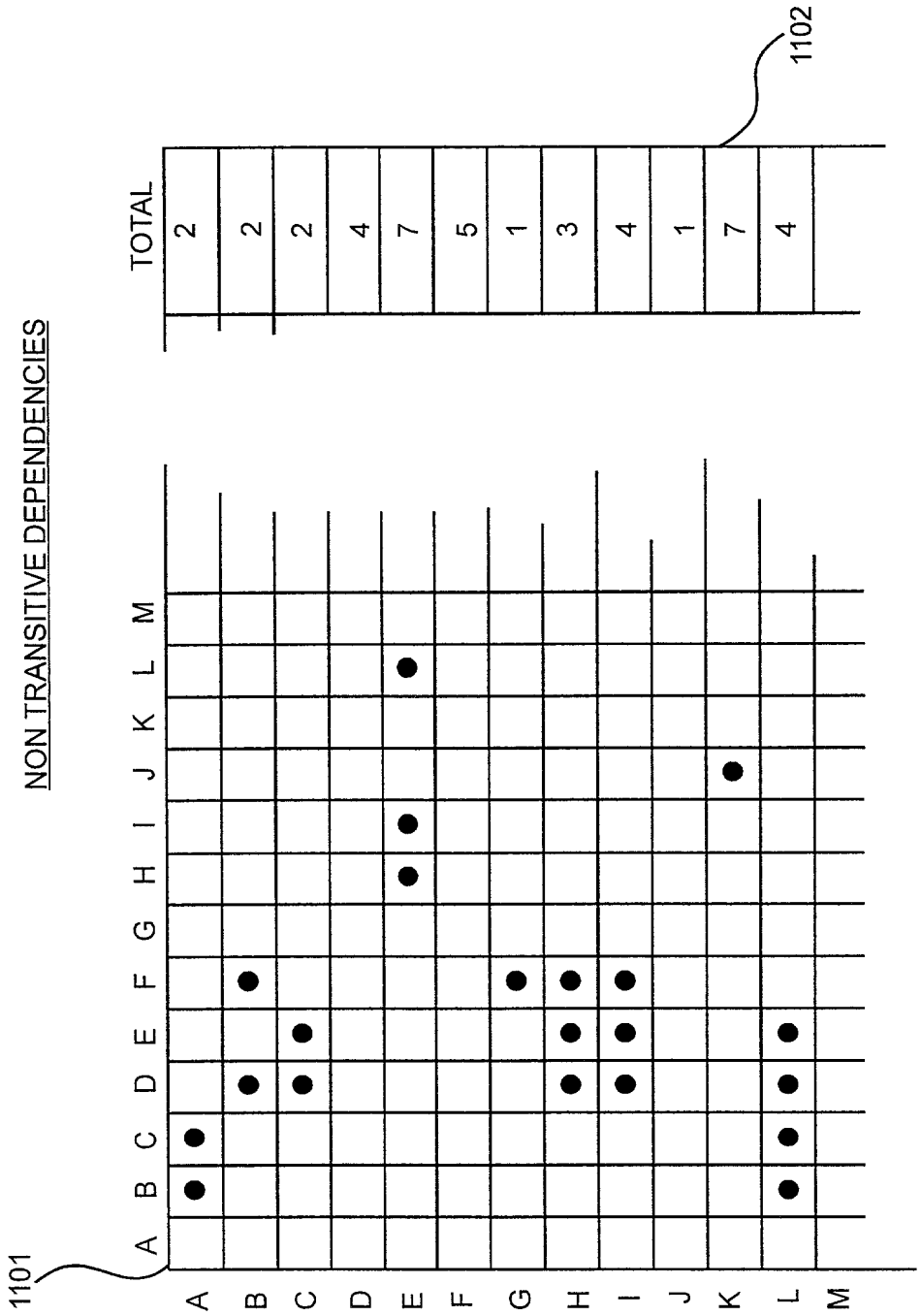

NON TRANSITIVE DEPENDENCIES
TOTAL
2
2
2
4
7
5
1
3
4
1
7
4
1102
1101
A B C D E F G H I J K L M
A
B
C
D
E
F
G
H
I
J
K
L
M

Figure 11

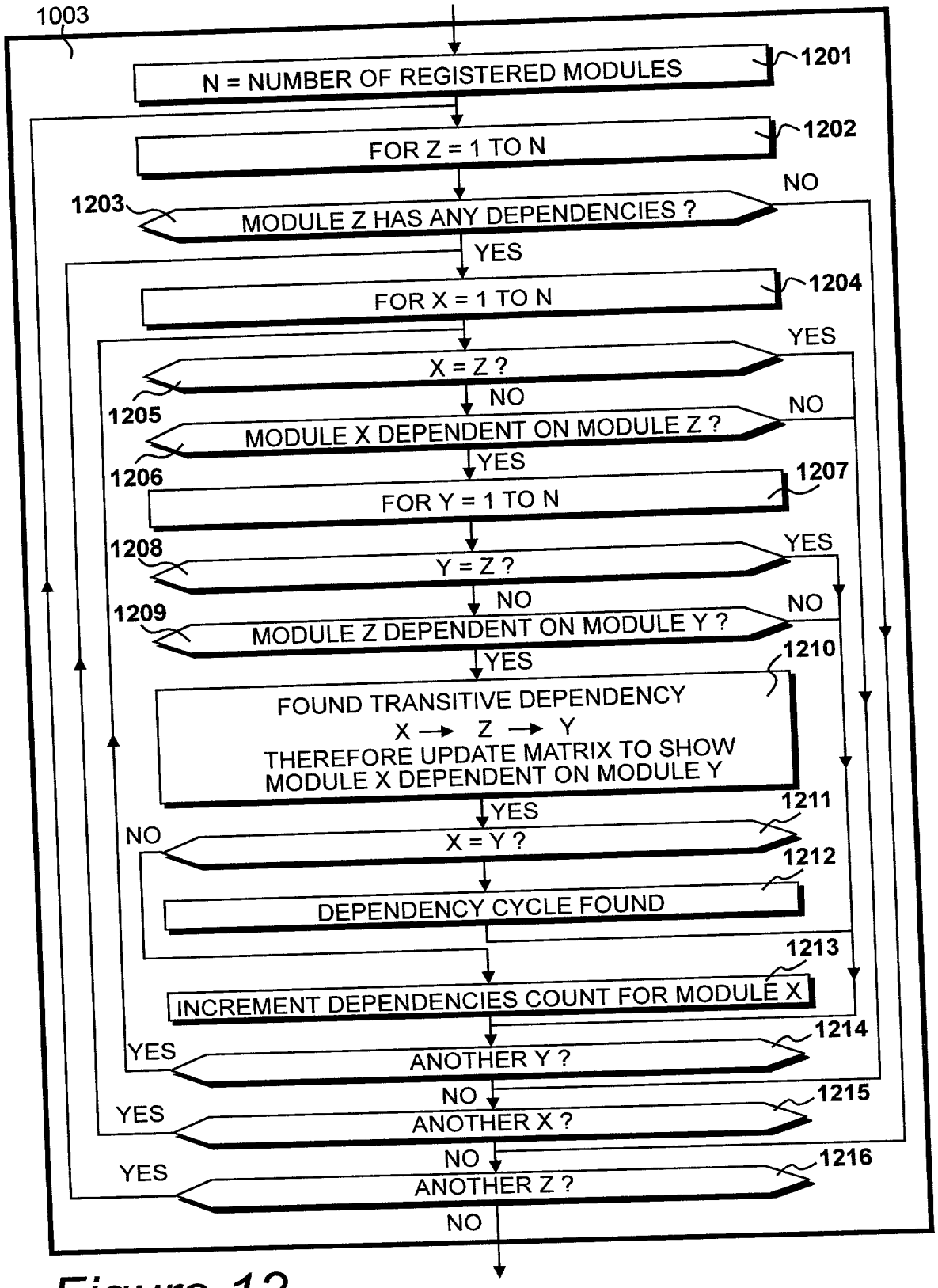

1003
N = NUMBER OF REGISTERED MODULES
1201
FOR Z = 1 TO N
1202
MODULE Z HAS ANY DEPENDENCIES ?
1203
NO
YES
FOR X = 1 TO N
1204
X = Z ?
YES
NO
MODULE X DEPENDENT ON MODULE Z ?
1205
NO
YES
FOR Y = 1 TO N
1206
1207
Y = Z ?
1208
YES
NO
MODULE Z DEPENDENT ON MODULE Y ?
1209
NO
YES
FOUND TRANSITIVE DEPENDENCY
X → Z → Y
THEREFORE UPDATE MATRIX TO SHOW
MODULE X DEPENDENT ON MODULE Y
1210
X = Y ?
YES
1211
NO
DEPENDENCY CYCLE FOUND
1212
INCREMENT DEPENDENCIES COUNT FOR MODULE X
1213
ANOTHER Y ?
1214
YES
NO
ANOTHER X ?
1215
YES
NO
ANOTHER Z ?
1216
YES
NO

Figure 12

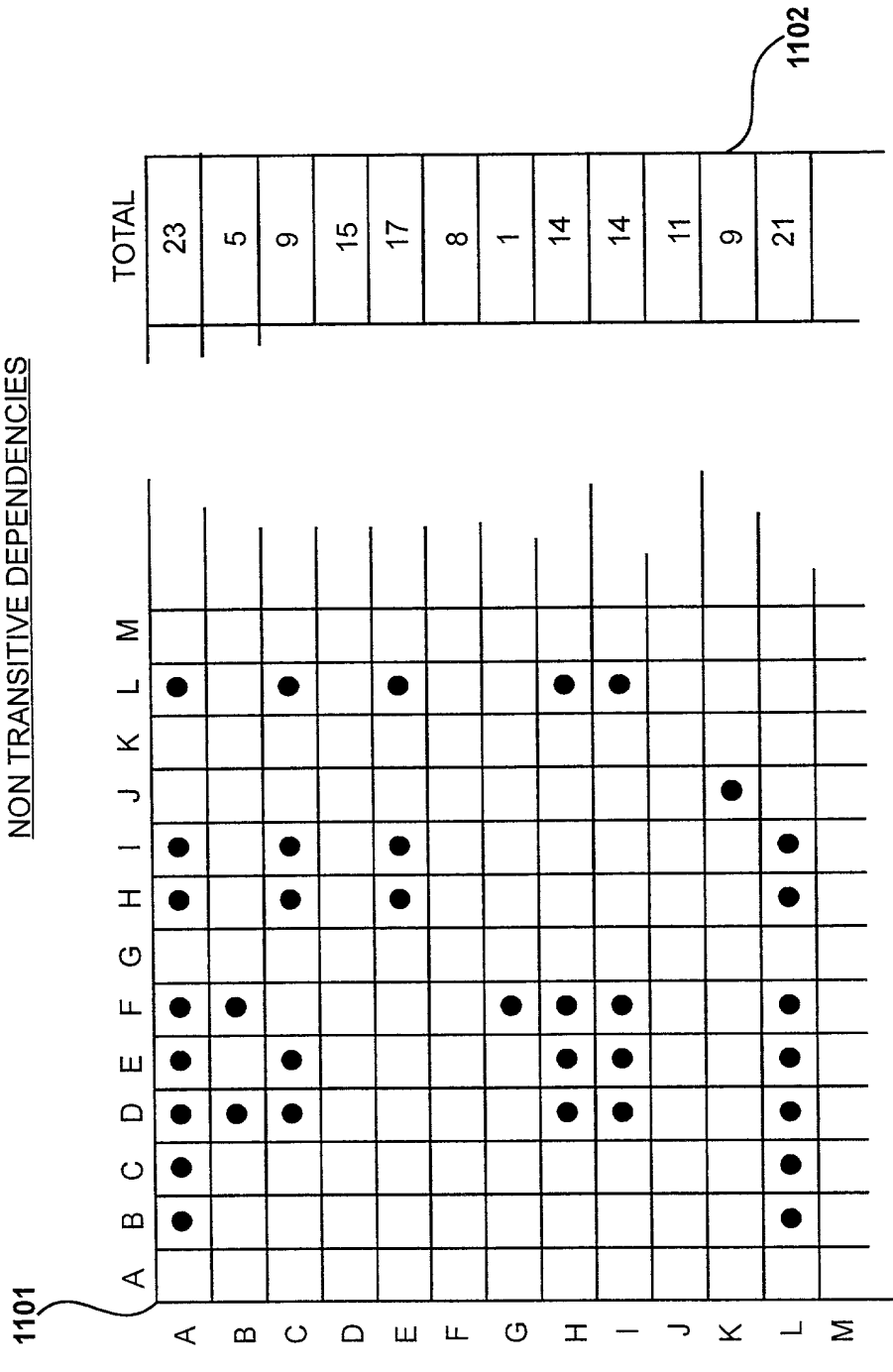

NON TRANSITIVE DEPENDENCIES
1101
A B C D E F G H I J K L M
A
B
C
D
E
F
G
H
I
J
K
L
M
TOTAL
23
5
9
15
17
8
1
14
14
11
9
21
1102

Figure 13

INITIALISATION SCHEDULE

MODULE	DEPENDENCIES	RANK
G	1	1
B	5	2
F	8	3
C	9	4
K	9	5
J	11	6
H	14	7
I	14	8
D	15	9
E	17	10
L	21	11
A	23	12

609

Figure 14

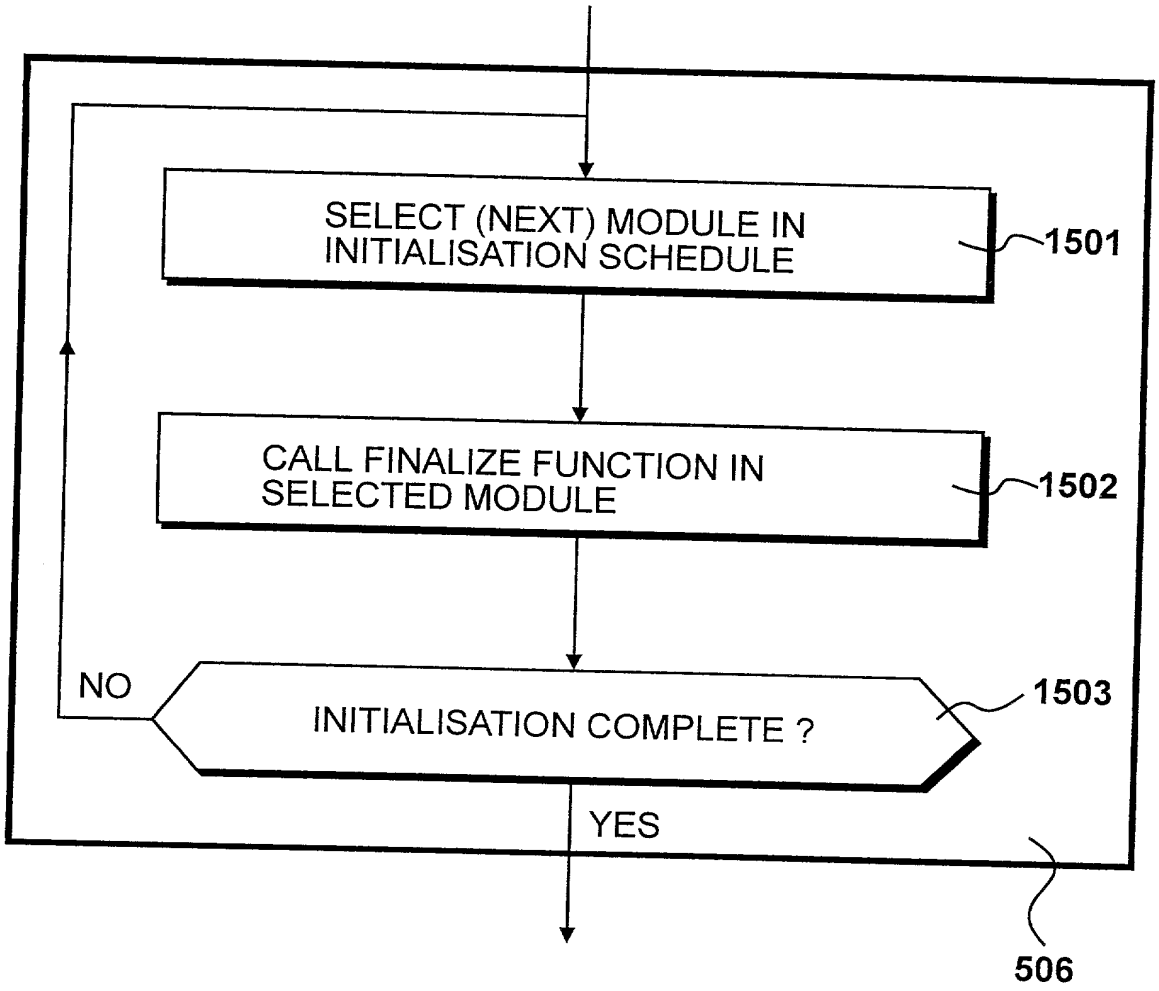

SELECT (NEXT) MODULE IN INITIALISATION SCHEDULE
1501
CALL FINALIZE FUNCTION IN SELECTED MODULE
1502
NO
INITIALISATION COMPLETE ?
1503
YES
506

Figure 15

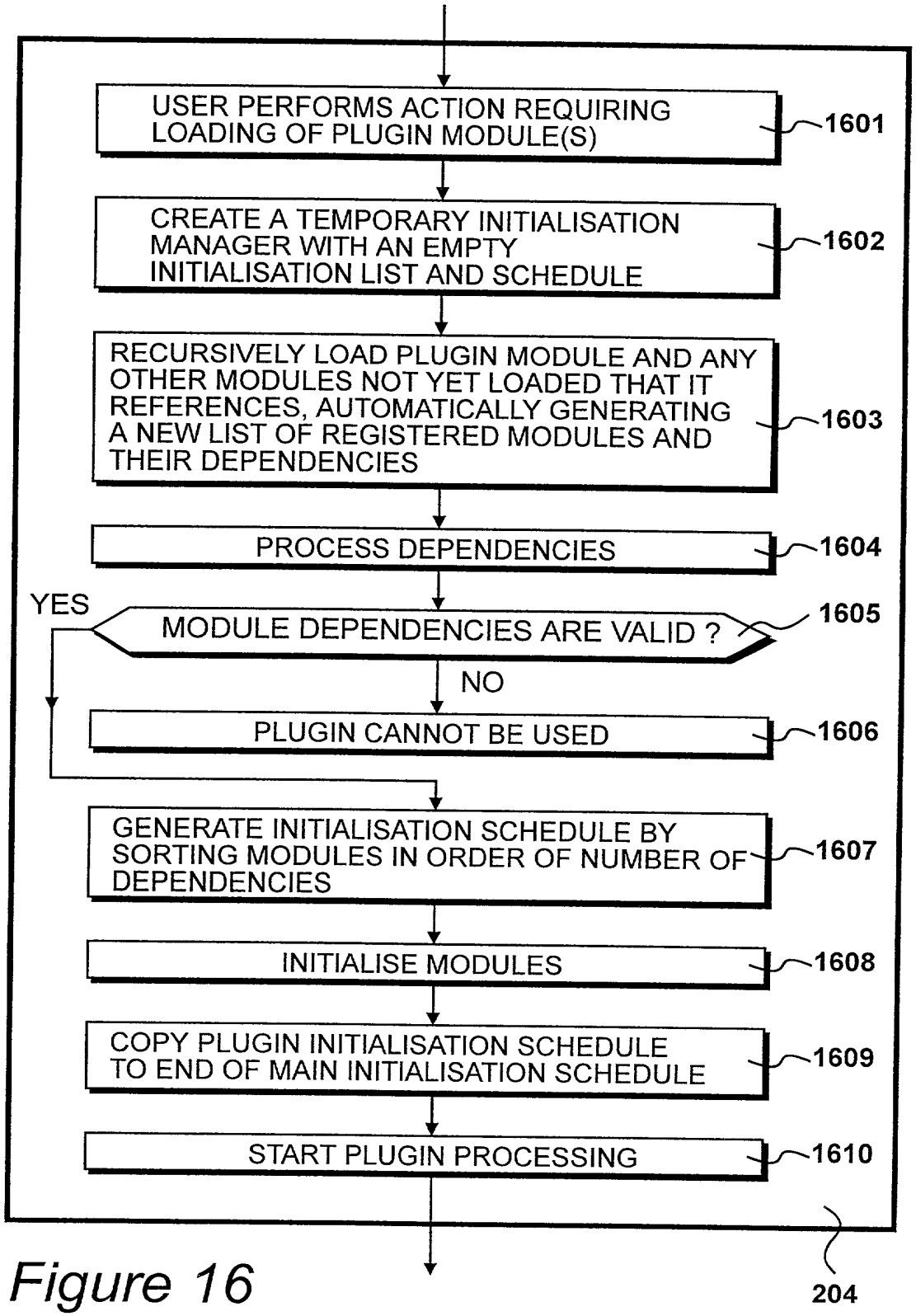

USER PERFORMS ACTION REQUIRING LOADING OF PLUGIN MODULE(S)
1601
CREATE A TEMPORARY INITIALISATION MANAGER WITH AN EMPTY INITIALISATION LIST AND SCHEDULE
1602
RECURSIVELY LOAD PLUGIN MODULE AND ANY OTHER MODULES NOT YET LOADED THAT IT REFERENCES, AUTOMATICALLY GENERATING A NEW LIST OF REGISTERED MODULES AND THEIR DEPENDENCIES
1603
PROCESS DEPENDENCIES
1604
YES
MODULE DEPENDENCIES ARE VALID ?
1605
NO
PLUGIN CANNOT BE USED
1606
GENERATE INITIALISATION SCHEDULE BY SORTING MODULES IN ORDER OF NUMBER OF DEPENDENCIES
1607
INITIALISE MODULES
1608
COPY PLUGIN INITIALISATION SCHEDULE TO END OF MAIN INITIALISATION SCHEDULE
1609
START PLUGIN PROCESSING
1610
204

Figure 16

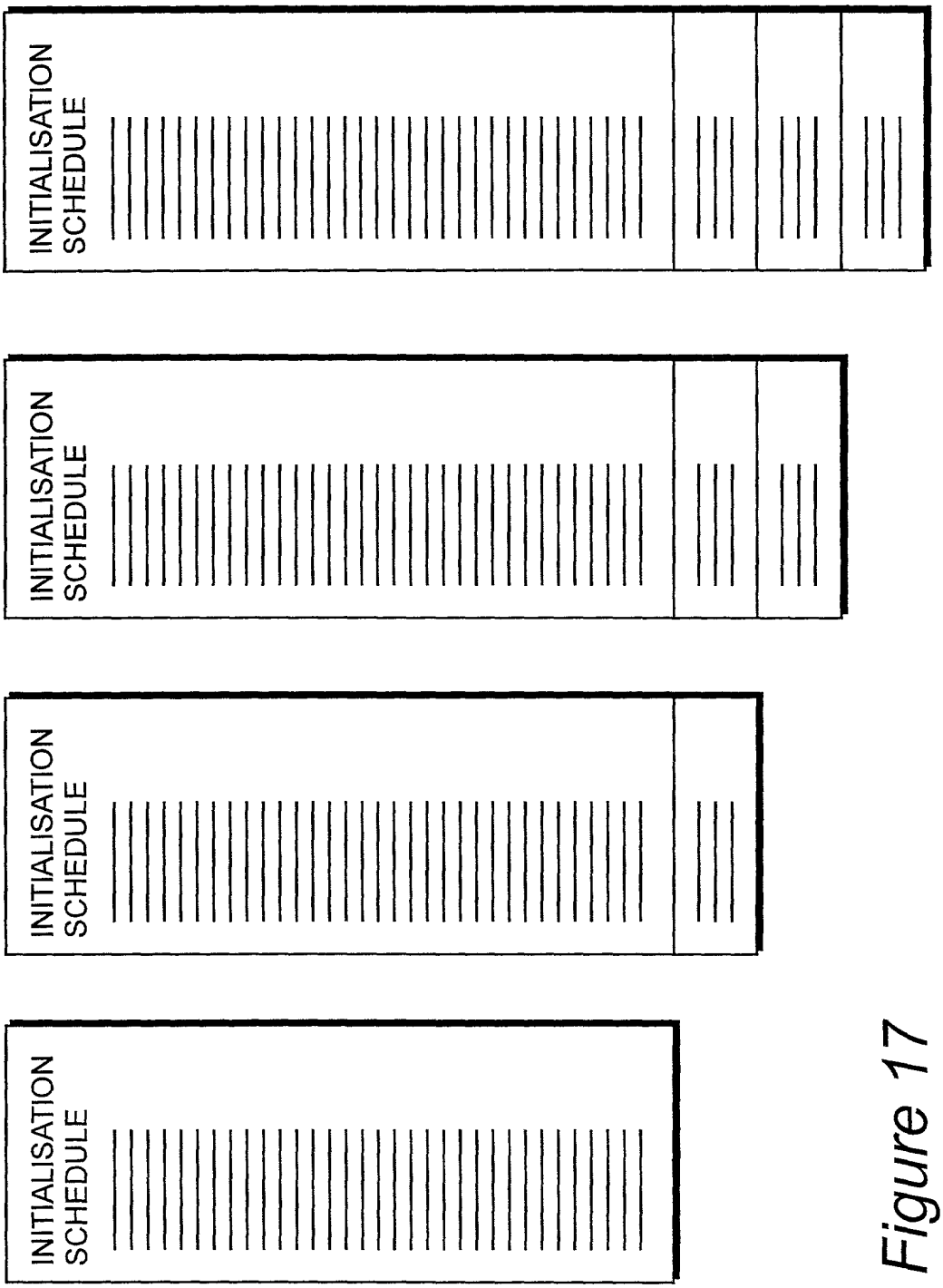

INITIALISATION SCHEDULE
INITIALISATION SCHEDULE
INITIALISATION SCHEDULE
INITIALISATION SCHEDULE

Figure 17

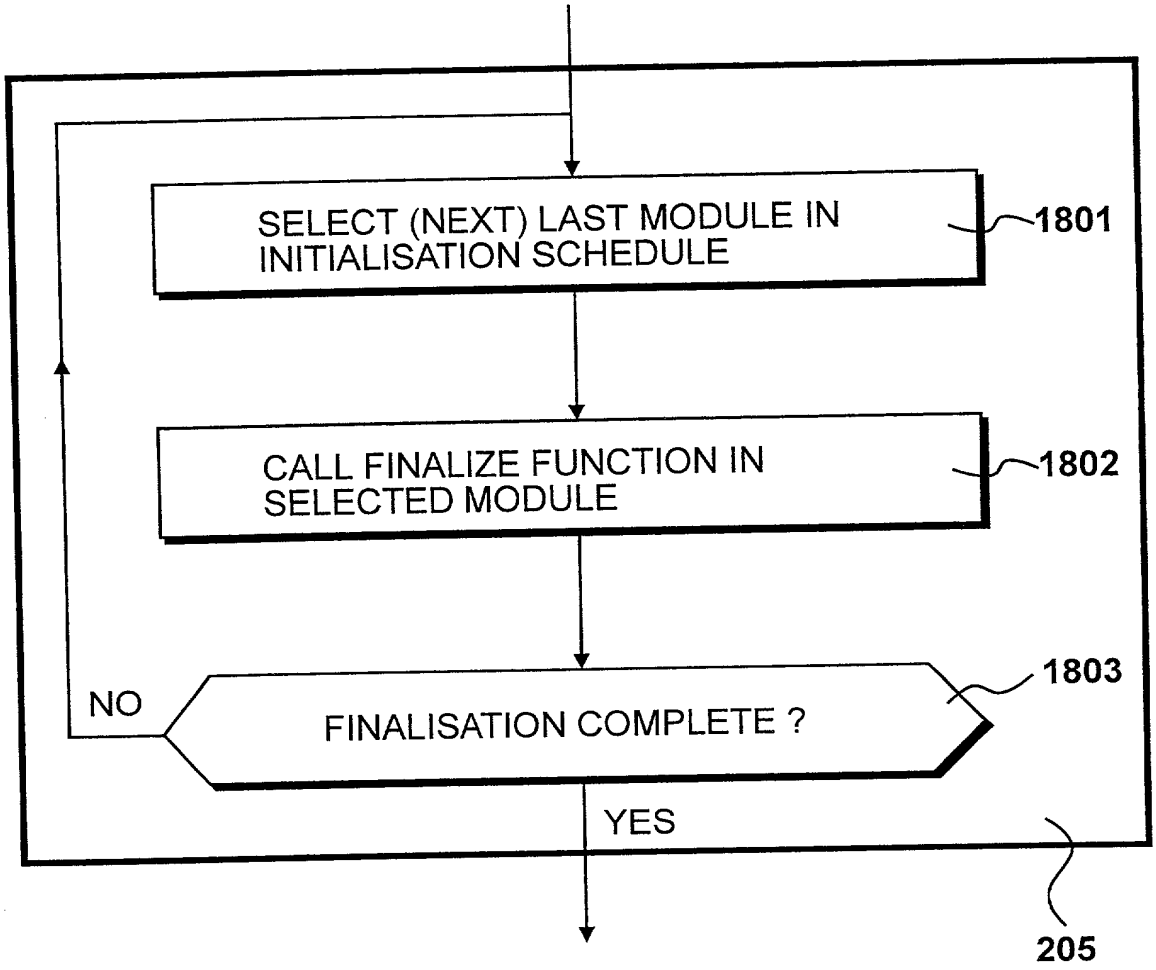

SELECT (NEXT) LAST MODULE IN INITIALISATION SCHEDULE
1801
CALL FINALIZE FUNCTION IN SELECTED MODULE
1802
1803
NO
FINALISATION COMPLETE ?
YES
205

Figure 18

```
// Filename: main.cpp

#include <fastmath.h>
#include <osal.h>
#include <initialize.h>

#include "main.h"

// Static objects

InInit<main> _instance; //registration object

// Constructor for registration object

InInit<main>::InInit () {

   registerDependency(typeInfo(InInit<fastmath>).typeID());
   registerDependency(typeInfo(InInit<osal>).typeID());
   registerDependency(typeInfo(InInit<initialize>).typeID());


}

main() {
{                                          1902
      InitGuard  ig;                                  1901

// the rest of the main function goes here
}                                          1903

// Other functions, include initialise() and finalise()
// go here ...
```

Figure 19

```
InitGuard : : InitGuard () {

     InInitManager.initialize () ;

}
```

Figure 20

```
void loadPlugin () {
         {  2101
             ReInitGuard rig;

             dlopen ("Plugin1");
             dlopen ("Plugin2");
         }  2102
}
```

Figure 21

```
ReInitGuard : : ReInitGuard () {

     ReInitManager.initialize () ;

}
```

Figure 22